Constructing Near Spanning Trees with Few Local Inspections*

Reut Levi† Guy Moshkovitz‡ Dana Ron§ Ronitt Rubinfeld¶ Asaf Shapira‖

Abstract

Constructing a spanning tree of a graph is one of the most basic tasks in graph theory. Motivated by several recent studies of local graph algorithms, we consider the following variant of this problem. Let G be a connected bounded-degree graph. Given an edge e in G we would like to decide whether e belongs to a connected subgraph G' consisting of $(1+\epsilon)n$ edges (for a prespecified constant $\epsilon > 0$), where the decision for different edges should be consistent with the same subgraph G'. Can this task be performed by inspecting only a *constant* number of edges in G? Our main results are:

- We show that if every t-vertex subgraph of G has expansion $1/(\log t)^{1+o(1)}$ then one can (deterministically) construct a sparse spanning subgraph G' of G using few inspections. To this end we analyze a "local" version of a famous minimum-weight spanning tree algorithm.
- We show that the above expansion requirement is sharp even when allowing randomization. To this end we construct a family of 3-regular graphs of high girth, in which every t-vertex subgraph has expansion $1/(\log t)^{1-o(1)}$.

1 Introduction

Given a graph G, one of the most basic tasks one would like to perform on G is to find a spanning tree of it or perhaps some other sparse spanning subgraph G'. This task can be easily accomplished using numerous well-known algorithms such as DFS (depth-first search), BFS (breadth-first search) and more. What all of these algorithms have in common is that in order to decide whether a given edge e belongs to the spanning subgraph G', one has to construct the entire spanning tree. Suppose however that one is not interested in constructing the entire spanning subgraph G', but rather to

*This work is partially based on an extended abstract that appeared in the proceedings of the eighteenth international workshop on randomization and computation (RANDOM) [21].

†École Normale Supèrieure and Universitè Paris Diderot, France. This work was done in part when Reut Levi was a PhD student at Tel Aviv University. Email: reuti.levi@gmail.com. Supported in part by ISF grants 246/08, 1147/09, 1536/14, NSF grants CCF-1217423, CCF-1065125, CCF-1420692 and by ANR grant RDAM

‡School of Mathematics, Tel Aviv University, Tel Aviv, Israel 69978. Email: guymosko@tau.ac.il. Supported in part by ISF grant 224/11.

§School of Electrical Engineering, Tel Aviv University. Tel Aviv 69978, Israel. Email: danar@eng.tau.ac.il. Supported in part by ISF grants 246/08 and 671/13.

¶CSAIL, MIT. Cambridge MA 02139, USA. School of Electrical Engineering, Tel Aviv University. Tel Aviv 69978, Israel. Email: ronitt@csail.mit.edu. Supported in part by NSF grants CCF-1217423, CCF-1065125, CCF-1420692 and by ISF grant 1536/14.

‖School of Mathematics, Tel Aviv University, Tel Aviv, Israel 69978. Email: asafico@tau.ac.il. Supported in part by ISF Grant 224/11 and a Marie-Curie CIG Grant 303320.

be able to "quickly" tell if a given edge e belongs to G' or not. By "quickly" we mean using a *constant* number of operations.

Such algorithms are of importance in distributed settings, where processors reside on the vertices of the graph and the goal is to select as few communication links (edges) as possible while maintaining connectivity. Another relevant setting is one in which the graph resides in a centralized database, but different, uncoordinated, servers have access to it, and are interested in only parts of a common sparse spanning subgraph. In both cases we would like the decision regarding any given edge to be made after inspecting only a very small portion of the whole graph, but all decisions must be consistent with the same spanning subgraph. Such algorithms belong to a growing family of local algorithms for solving classical problems in graph theory. We elaborate on relevant related works in Subsection 1.1.

Let us make a simple observation regarding the task of locally constructing a spanning subgraph. Note that if one insists on locally constructing a spanning *tree* G', then it is easy to see that the task cannot be performed in general without inspecting almost all of G; that is, this task cannot be achieved using a *constant* number of queries to G. To see this, observe that if G consists of a single path, then the algorithm must answer positively on all edges, while if G consists of a single cycle then the algorithm must answer negatively on one edge. However, the two cases cannot be distinguished without inspecting a linear number of edges.

So suppose we allow the algorithm some slackness, and rather than requiring that G' be a tree, only require that it be relatively sparse, i.e., contain at most $(1+\epsilon)n$ edges. Summarizing, the question is then, given $\epsilon > 0$, for which graphs G can we locally construct a spanning subgraph G' consisting of $(1+\epsilon)n$ edges, such that given an edge $e \in E(G)$ one can determine if $e \in G'$ using a constant (that may depend on ϵ but not on n) number of queries to G?

Our main result in this paper, stated informally as Theorem 1 below, shows that the answer to the above question is given by a certain variant of graph expansion, which we now turn to define. For a graph G and a subset $S \subseteq V(G)$, we write $\partial_G(S)$ for the set of edges of G with precisely one endpoint in S. We write ϕ_G for the (edge) *expansion* of G, that is, $\phi_G = \min_S |\partial_G(S)| / |S|$ where the minimum is taken over all $S \subseteq V(G)$ of size $1 \leq |S| \leq |V(G)|/2$. Note that a graph may have small expansion yet contain (large) subgraphs with large expansion. For example, a vertex-disjoint union of cliques has expansion 0, yet it contains complete graphs that have the largest expansion possible (for graphs of their order). Let us thus say that a graph is *f-non-expanding* if every t-vertex subgraph H satisfies $\phi_H \leq f(t)$ (we assume $t > 2$).

Our main result in this paper can be informally stated as follows.

Theorem 1 (Informal Statement). *We have the following dichotomy:*

- *If G is f-non-expanding for $f(t) \ll 1/\log t$ then one can locally construct a sparse spanning subgraph of G. The algorithm is deterministic.*
- *There is a family of 3-regular graphs G_n that are (roughly) $1/\log t$-non-expanding so that every (possibly randomized) local algorithm for constructing a sparse spanning subgraph of G_n must accept every edge of G_n.*

We refer the reader to Definition 1 for the precise definition of what it means to locally construct a sparse spanning subgraph, and to Theorems 2 and 3 for the precise statements of the two assertions in Theorem 1.

We note that there are numerous families of graphs that satisfy the condition in the first item of Theorem 1. For example, it follows from the planar separator theorem of Lipton and Tarjan [23] and its extension by Alon, Seymour and Thomas [3] that planar graphs (and more generally, H-minor-free graphs) of bounded degree satisfy the condition of the first item. Also, observe that since the graphs G_n in the second item of Theorem 1 have $3n/2$ edges, there is no algorithm that can locally construct a spanning subgraph of G_n with $(1+\epsilon)n$ edges for $\epsilon < 1/2$.

We make two comments regarding the results which appeared in the preliminary conference version of this paper [21]. First, it was shown in [21] that there are graphs such that any algorithm has to inspect $\Omega(\sqrt{n})$ edges in order to decide whether a given edge belongs to a spanning subgraph G' containing $(1+\epsilon)n$ edges, for a constant ϵ. However, those graphs resulted from random graphs, which have expansion $\Theta(1)$, and so could not be used in order to show that the non-expansion requirement given in the first item of Theorem 3 cannot be relaxed. Second, it was shown in [21] that for certain families of graphs, one can solve the sparse spanning subgraph problem in time $O(\sqrt{n})$. It is an interesting open problem to decide whether this can be extended to hold for all bounded-degree graphs. In fact, it would even be interesting to show that for any bounded-degree graph G, one can find a sparse spanning subgraph using $o(n)$ queries[1].

1.1 Related work

As is evident from the above description of the problem, the model we study here is similar to both classical models, such as distributed and parallel computation, and to more recent ones. In what follows, we describe these models and some related results, so as to provide a broad context for our work.

1.1.1 Local algorithms for other graph problems

The model of *local computation algorithms* as considered in this work, was defined by Rubinfeld et al. [36] (see also Alon et al. [2]). Such algorithms for maximal independent set, hypergraph coloring, k-CNF and maximum matching are given in [36, 2, 25, 26]. This model generalizes other models that have been studied in various contexts, including locally decodable codes (e.g., [24]), local decompression [14], and local filters/reconstructors [1, 37, 9, 18, 17, 12]. Local computation algorithms that give approximate solutions for various optimization problems on graphs, including vertex cover, maximal matching, and other packing and covering problems, can also be derived from sublinear time algorithms for parameter estimation [33, 27, 31, 15, 40].

The model of local computation is related to several other models, including property testing and online algorithms. To give a notable example, Mansour et al. [25] proposed a general scheme for converting a large family of online algorithms into local computation algorithms, consequently, improving the complexity of hypergraph 2-coloring and k-CNF in the local computation model.

In the related field of local reconstructors, Campagna et al. [10] study the property of connectivity. Namely, under the promise that the input graph is almost connected, their reconstructor provides oracle access to the adjacency matrix of a connected graph which is close to the input graph. We emphasize that our model is different from theirs, in that they allow the addition of new

[1]Note that if we are allowed to make $\Theta(n)$ queries, then we can just use the standard BFS or DFS algorithms, which find the entire spanning tree of G.

edges to the graph, whereas our algorithms must provide spanning graphs whose edges are present in the original input graph.

1.1.2 Distributed and parallel algorithms

The name *local algorithms* is also used in the distributed context [28, 30, 22]. As observed by Parnas and Ron [33], local distributed algorithms can be used to obtain local computation algorithms as defined in this work, by simply emulating the distributed algorithm on a sufficiently large subgraph of the graph G. However, while the main complexity measure in the distributed setting is the number of rounds (where it is usually assumed that each message is of length $O(\log n)$), our main complexity measure is the number of queries performed on the graph G. By this standard reduction, the bound on the number of queries (and hence running time) depends on the size of the queried subgraph and may grow exponentially with the number of rounds. Therefore, this reduction gives meaningful results only when the number of rounds is significantly smaller than the diameter of the graph.

While the problem of computing a spanning graph has not been studied in the distributed model, the problem of computing a minimum-weight spanning tree is a central one in this model. Kutten and Peleg [20] provided an algorithm that works in $O(\sqrt{n}\log^* n + D)$ rounds, where D denotes the diameter of the graph. Their result is nearly optimal in terms of the complexity in n, as shown by Peleg and Rubinovich [34] who provided a lower bound of $\Omega(\sqrt{n}/\log n)$ rounds (when the length of the messages must be bounded).

Another problem studied in the distributed setting that is related to the one studied in this paper, is finding a sparse spanner. The requirement for spanners is much stronger since the distortion of the distance should be as small as possible. Thus, to achieve this property, it is usually the case that the number of edges of the spanner is super-linear in n. Pettie [35] was the first to provide a distributed algorithm for finding a low distortion spanner with $O(n)$ edges without requiring messages of unbounded length or $O(D)$ rounds. The number of rounds of his algorithm is $\log^{1+o(1)} n$. Hence, the standard reduction of [33] yields a local algorithm with a trivial linear bound on the query complexity.

The problems of computing a spanning tree and a minimum weight spanning tree were studied extensively in the parallel computing model as well (see, e.g., [7], and the references therein). However, these parallel algorithms have time complexity which is at least logarithmic in n and therefore do not yield an efficient algorithm in the local computation model. See [36, 2] for further discussion on the relationship between the ability to construct local computation algorithms and the parallel complexity of a problem.

1.1.3 Local cluster algorithms

Local algorithms for graph theoretic problems have also been given for PageRank computations on the web graph [16, 8, 38, 5, 4]. Local graph partitioning algorithms have been presented in [39, 5, 6, 41, 32], which find subsets of vertices whose internal connections are significantly richer than their external connections in time that depends on the size of the cluster that they output. For instance, Andersen and Peres [6] provide an algorithm which, given a starting vertex v, finds a cluster of v of small conductance, whose complexity depends on the volume of the cluster it outputs but has only polylogarithmic dependence in the size of the graph. However, even when the size of the cluster

is guaranteed to be small, it is not obvious how to use these algorithms in the local computation setting where the cluster decompositions must be consistent among queries to all vertices.

1.1.4 Other related sublinear-time approximation algorithms for graphs

The problem of estimating the weight of a minimum-weight spanning tree in sublinear time was considered by Chazelle, Rubinfeld and Trevisan [11]. They describe an algorithm whose running time depends on the approximation parameter, the average degree and the range of the weights, but does not directly depend on the number of vertices.

1.2 Organization

The rest of the paper is organized as follows. In Section 2 we formally define the local sparse spanning subgraph problem which we consider in this paper. In Section 3 we prove the first item of Theorem 1, which is formally stated as Theorem 2. The proof of this part has two main steps. In the first one, we show that if G is f-non-expanding with $f \ll 1/\log t$ then one can remove from G only a relatively small number of edges and thus partition it into connected components of size $O(1)$ each. We then show that if a graph can be so partitioned, then one can solve on it the local spanning subgraph problem by executing a "localized" version of Kruskal's [19] famous algorithm for finding minimum-weight spanning tress[2].

The proof of the second paper of Theorem 1, which is the more challenging part of this paper, is given in Section 4 and formally stated as Theorem 3. It establishes that the $1/\log t$-non-expansion requirement from the first item of Theorem 1 is essentially tight. What we show is that there are graphs which are (about) $1/\log t$-non-expanding, and have the property that any local algorithm for constructing a spanning subgraph using a constant number of queries must accept *every* edge of the graph. To prove this result we describe a construction of certain extremal graphs that might be of independent interest. These are 3-regular graphs, that on one hand have unbounded *girth*[3], but on the other hand are (about) $1/\log t$-non-expanding.

We make no serious attempt to optimize the constants obtained in the various statements. In fact, the f-non-expansion requirements in our upper and lower bound results (Theorems 2 and 3), which are about $(1/\log t)(1/\log\log t)^2$ and $(1/\log t)(\log\log t)^2$ respectively, can each be improved by replacing the $(\log\log t)^2$ term by $(\log\log t)^{1+o(1)}$. We opted for proving our results with the slightly weaker bounds in order to simplify the presentation. We henceforth write $\log(\cdot)$ for $\log_2(\cdot)$.

2 Preliminaries

Let us now give the precise definition of the algorithmic problem we are addressing in this paper. As in most cases where one tries to design a local/distributed/sublinear algorithm, we will assume that the input graph is given via an *oracle access to its incidence-list representation*, meaning the following: First, we assume that the input graph $G = (V, E)$ is given via incidence-lists representation, that is, for each vertex $v \in V(G)$, there is an ordered list of its neighbors in G. Second, the

[2]Recall that if G is a graph with weights assigned to its edges, then Kruskal's algorithm finds a spanning tree of minimal *total* weight

[3]As usual, the girth of a graph is the minimum length of a cycle in it.

algorithm is supplied with integers n and d, that represent the number of vertices, and an upper bound on the degrees of vertices of G. Finally, given a pair (v, i) with $1 \leq v \leq n$ and $1 \leq i \leq d$, the oracle either returns the i^{th} neighbour of v (in the incidence list representation) or an indication that v has less than i neighbours. We will assume that each vertex v has an id, $id(v)$, where there is a full order over the ids. We will think of the ids of vertices in the graphs simply as the integers $\{1, \ldots, n\}$. We now turn to formally define the algorithmic problem we consider in this paper.

Definition 1. *An algorithm $\mathcal{A}$ is an* (ϵ, q)-local sparse spanning graph algorithm *if, given $n, d \geq 1$ and oracle access to the incidence-lists representation of a connected graph $G = (V, E)$ on n vertices and degree at most d, it provides query access to a subgraph $G' = (V, E')$ of G such that:*

i. G' is connected.

ii. $|E'| < (1 + \epsilon) \cdot n$ with probability at least $2/3$ (over the internal randomness of $\mathcal{A}$).

iii. E' is determined by G and the internal randomness of $\mathcal{A}$.

iv. $\mathcal{A}$ makes at most q queries to G.

By "providing query access to G'" we mean that on input $(u, v) \in E$, $\mathcal{A}$ returns whether $(u, v) \in E'$ and for any sequence of queries, $\mathcal{A}$ answers consistently with the same G'.

An algorithm $\mathcal{A}$ is an (ϵ, q)-local sparse spanning graph algorithm for a family of graphs $\mathcal{C}$ *if the above conditions hold, provided that the input graph G belongs to $\mathcal{C}$.*

We note that the choice of the required success probability being $2/3$ is of course arbitrary and can be replaced by any probability smaller than 1. Having said this, let us stress that the positive results we obtain here (i.e., the algorithmic results) in Theorem 2 are deterministic (i.e., hold with probability 1), whereas our lower bound in Theorem 3 holds for *any* positive success probability. We also note that even though Definition 1 considers only the number of queries performed by the algorithm, our algorithm in Theorem 2 runs in time polynomial in the number of queries, and in particular, independent of n.

We are interested in local algorithms that have query complexity which is independent of n, namely, that perform a constant number of queries to the graph (for each edge they are queried on) and whose running time (per queried edge) is small as well. In the next section, we show that the family of graphs that are f-non-expanding with $f \ll 1/\log t$ have a local sparse spanning graph algorithm. In the following section, we will show that one cannot prove such a result when f is only slightly larger.

3 Upper bound

In this section we prove the following theorem, which formalizes the first assertion of Theorem 1.

Theorem 2. *For every C there is a function $q : \mathbb{R}_+ \times \mathbb{N} \to \mathbb{N}$ so that for every $\epsilon > 0$ there is an $(\epsilon, q(\epsilon, d))$-local sparse spanning graph algorithm for the family of f-non-expanding graphs with*

$$f(x) = \frac{C}{\log x \cdot (\log \log x)^2}\ , \tag{1}$$

where d is the input degree-bound. Furthermore, the algorithm is deterministic.

3.1 Decomposition of non-expanding graphs

The first step in the proof of Theorem 2 is a decomposition result stated in Lemma 1 below. It shows that if G is f-non-expanding, with f as in Equation (1), then G can be decomposed into connected components of bounded size by removing only ϵn edges. This extends a result of [13] that applies for somewhat larger f. As mentioned earlier, there are many families of graphs which are f-non expanding with f as in Equation (1). For example, planar graphs of bounded degree are f-non-expanding with $f = O(1/\sqrt{x})$ by the famous planar separator theorem of Lipton and Tarjan [23]. More generally, a result of Alon, Seymour and Thomas [3] implies that for any fixed H, the family of H-minor-free graphs of bounded degree is f-non-expanding with $f = O(1/\sqrt{x})$. Hence, Lemma 1 applies to these families of graphs in particular. We note that the reason why the bound in Lemma 1 is doubly exponential in ϵ is that we insist on assuming that f is very close to the threshold of $1/\log x$ (which by Theorem 3 is essentially tight). For example, the details of the proof of Lemma 1 show that if $f = x^{-c}$ for some $0 < c < 0$ (as is the case with planar graphs, say), then the bound can be improved to polynomial in $1/\epsilon$. We note that in such cases we can also set $k = \text{poly}(1/\epsilon)$ in step 1 of our algorithm (Algorithm 1 below), thus obtaining a much more efficient algorithm.

Lemma 1. *If G is an n-vertex f-non-expanding graph with $f(x) = C/\log x(\log\log x)^2$, then one can remove ϵn edges from G so that each connected component of the remaining graph is of size at most $2^{2^{2(C/\epsilon)+3}}$.*

Proof: First, we claim that any f-non-expanding n-vertex graph $G = (V, E)$ has a subset $S \subset V(G)$ of size $n/3 \leq |S| \leq (2/3)n$ and expansion $\phi_G(S) \stackrel{\text{def}}{=} |\partial_G(S)|\,/\,|S| \leq f(n/3)$. For the proof we iteratively construct subsets $S_1, \ldots, S_k \subseteq V(G)$ as follows. To obtain S_i, we consider the induced subgraph $G_i = G[V \setminus \bigcup_{j=1}^{i-1} S_j]$ and let $S_i \subseteq V(G_i)$ satisfy $|S_i| \leq n_i/2$ and $\phi_{G_i}(S_i) \leq f(n_i)$, where $n_i = |V(G_i)|$. We stop once $S \stackrel{\text{def}}{=} \bigcup_{i=1}^{k} S_i$ is of size $|S| \geq n/3$. Note that

$$|S| \leq \sum_{i=1}^{k-1} |S_i| + n_k/2 = (n + \sum_{i=1}^{k-1} |S_i|)/2 \leq 2n/3 \; .$$

It remains to bound $\phi_G(S)$. Observe that every edge in the edge boundary $\partial_G(S)$ is a member of some edge boundary $\partial_{G_i}(S_i)$. Hence,

$$\frac{|\partial_G(S)|}{|S|} \leq \frac{\sum_{i=1}^{k} |\partial_{G_i}(S_i)|}{|S|} = \sum_{i=1}^{k} \frac{|S_i|}{|S|} \phi_{G_i}(S_i) \leq \max_{1 \leq i \leq k} \phi_{G_i}(S_i) \leq \max_{1 \leq i \leq k} f(n_i) = f(n_k) \leq f(n/3) \; ,$$

where in the last inequality we used the fact that $n_k \geq n - |S| \geq n/3$. This proves our claim.

Fix an integer $k \geq 50$ and let $G = (V, E)$ be any f-non-expanding graph on $n \geq k/3$ vertices. Consider the following process; take any subset $S \subset V$ of size $n/3 \leq |S| \leq n/2$ and expansion $\phi_G(S) \leq 2f(n/3)$ (whose existence follows from the claim above), remove all its outgoing edges and proceed recursively on the two induced subgraphs $G[S]$ and $G[V \setminus S]$, which are f-non-expanding as well. The recursion stops whenever we reach a graph on at most k vertices. It is clear that at the end of this process, the edges removed from G leave a graph whose connected components have at most k vertices each. Let $r_k(G)$ be the number of edges removed by the above process.

We will shortly prove that if G has n vertices, then $r_k(G) \leq Cn/\ln\ln(k/3)$. Hence, setting $k = 2^{2^{2(C/\epsilon)+3}} \geq \max\{50, 3 \cdot e^{e^{C/\epsilon}}\}$ enables us to remove no more than ϵn edges and break G into connected components of size at most k, thus proving the lemma.

In order to facilitate an inductive proof, it will be more convenient to prove the following slightly stronger claim:

$$r_k(G) \leq \beta(n) \stackrel{\text{def}}{=} \frac{Cn}{\ln\ln(k/3)} - \frac{Cn}{\ln\ln n} \,. \tag{2}$$

Set $h(x) = x/\ln\ln x$ and $f^*(x) = f(x)/C = (\log x)^{-1}(\log\log x)^{-2}$. First, we establish some properties of h. It is easy to verify that the derivative of h is $h'(x) = (\ln\ln x)^{-1} - (\ln x)^{-1}(\ln\ln x)^{-2}$, and moreover, $h''(x) \leq 0$ for $x \geq 20$. It follows that for every $n \geq 50$ and $n/3 \leq s \leq n/2$ we have

$$h(n) - h(n-s) \leq s \cdot h'(n-s) \leq h(s) - s \cdot 2f^*(n/3) \tag{3}$$

where in the first inequality we used the concavity of h on the interval $(20, \infty)$, and in the second inequality we used the fact that $n \geq 50$ and $n/3 \leq s \leq n/2$ and that in this range

$$(\log(n/3)(\log\log(n/3))^2/2 \geq \ln(2n/3)(\ln\ln(2n/3))^2 \geq \ln(n-s)(\ln\ln(n-s))^2 \,.$$

We prove Equation (2) by induction on n. For the base case(s) where $(k/3 \leq)\ n \leq k$ we have that $\beta(n) \geq \beta(k/3) = 0 = r_k(G)$, as needed. For the induction step we have

$$\begin{aligned}
r_k(G) &\leq \max_{\substack{S \subset V: \\ n/3 \leq |S| \leq n/2}} |S| \cdot 2f(n/3) + r_k(G[S]) + r_k(G[V \setminus S]) \\
&\leq \max_{n/3 \leq s \leq n/2} s \cdot 2f(n/3) + \beta(s) + \beta(n-s) \\
&= C\Big(n/\ln\ln(k/3) + \max_{n/3 \leq s \leq n/2} s \cdot 2f^*(n/3) - h(s) - h(n-s)\Big) \\
&\leq C\Big(n/\ln\ln(k/3) - h(n)\Big) = \beta(n)
\end{aligned}$$

where the first inequality follows from the definition of the process described in the second paragraph of the proof, the second inequality follows from the induction hypothesis since $k/3 \leq s, n-s \leq n-1$, and in the third inequality we used (3) since $n \geq k \geq 50$. This completes the proof of Equation (2). ■

3.2 The algorithm

The algorithm we design in order to prove Theorem 2 is based on Kruskal's minimum-weight spanning tree algorithm [19]. The idea is to assign weights to the edges of the graph in a way that will determine some fixed spanning tree T. The algorithm will always accept the edges of T but will also accept a few other edges. We will pick the weights of the edges in a way that will make it possible to determine the edges of a sparse spanning subgraph in a "local" fashion.

Recall that Kruskal's algorithm for finding a minimum-weight spanning tree in a weighted connected graph works as follows. First it sorts the edges of the graph $e_1, \ldots e_m$ from minimum to maximum weight (breaking ties arbitrarily). It then goes over the edges in this order, and adds e_i to the spanning tree if and only if it does not close a cycle with the previously selected edges. Put differently:

Fact 1. *Edge e is picked by Kruskal's algorithm if and only if for any cycle C of G containing e, the edge e does not have the largest weight among the edges of C.*

It is well known (and easy to verify) that if the weights of the edges are distinct, then there is a single minimum-weight spanning tree in the graph. For an unweighted graph G, consider the order defined over its edges by the order of the ids of the vertices. Namely, we define a ranking r of the edges as follows: $r(u,v) < r(u',v')$ if and only if $\min\{id(u), id(v)\} < \min\{id(u'), id(v')\}$ or $\min\{id(u), id(v)\} = \min\{id(u'), id(v')\}$ and $\max\{id(u), id(v)\} < \max\{id(u'), id(v')\}$. If we run Kruskal's algorithm using the rank r as the weight function (where there is a single ordering of the edges), then we obtain a (well-defined) spanning tree of G.

While the local algorithm described next (Algorithm 1) is based on the aforementioned global algorithm, it does not exactly emulate it, but rather emulates a certain *relaxed* version of it which can be executed *locally*. In particular, it will answer YES for every edge selected by the global algorithm (ensuring connectivity), but may answer YES also on edges not selected by the global algorithm. We will thus need to show that it does not answer YES on too many edges that are not selected by the global algorithm.

In the description and analysis of the algorithm we will use the following standard notation; for a vertex $v \in V$ and an integer k, we denote by $C_k(v, G)$ the subgraph of G induced by the set of vertices at distance at most k from v.

Algorithm 1 (Kruskal-based Algorithm)

(The algorithm works for some fixed $\epsilon > 0$.)
Input: $n, d \geq 1$, query access to a graph G on n vertices and degree at most d.
Queried edge: $(x, y) \in E(G)$.

1. Set $k = 2^{2^{2(C/\epsilon)+3}}$.
2. Perform a BFS to depth k from x, thus obtaining the subgraph $C_k(x, G)$.
3. If (x, y) is the edge with largest rank on some cycle in $C_k(x, G)$ then answer NO; Otherwise, answer YES.

Proof of Theorem 2: We will show that if $G = (V, E)$ is $C/\log x(\log\log x)^2$-non-expanding then Algorithm 1 is an $(\epsilon, q(\epsilon, d))$-local sparse spanning subgraph algorithm, where $q(\epsilon, d) = d^{k+1}$ with k being the constant from step 1 of the algorithm. By the description of Algorithm 1 it directly follows that the algorithm is deterministic and that its answers are consistent with a connected subgraph G'. Indeed, if T is the tree returned by Kruskal's algorithm, then Fact 1 and step 3 of Algorithm 1 guarantee that each edge of T will be accepted by Algorithm 1. Observe that the number of queries to G performed by Algorithm 1 is at most d^{k+1}. We now complete the proof by showing that the algorithm returns YES on fewer than $(1+\epsilon)n$ edges.

Let R be a set of at most ϵn edges whose removal disconnects G into connected components of size at most k. The existence of such a set is guaranteed by Lemma 1. Let G_R be the graph obtained by removing R from G; that is, $G_R = (V, E \setminus R)$. We note (crucially) that while the analysis of the algorithm uses properties of G_R, the algorithm does not actually compute R. We

will now show that G' does not contain a cycle of G_R. Since $|R| \leq \epsilon n$, this means that G' has fewer than $(1+\epsilon)n$ edges.

Let σ be a cycle in G_R. Suppose (w, v) is the edge of σ with the largest rank. Since the connected components of G_R are of size at most k, we infer that σ has at most k vertices, implying that $C_k(w, G)$ contains σ. It follows that on query (w, v) the algorithm will return NO. Thus, G' does not contain σ. ■

4 Lower bound

The next theorem shows that there are graphs for which any local sparse spanning graph algorithm must perform a number of queries that grows with n, yet these graphs are f-non-expanding with $f(x)$ only slightly larger than $1/\log x$. This is essentially the best one can hope for in light of Theorem 2.

Theorem 3. *For infinitely many n, there is an f-non-expanding n-vertex graph G with*

$$f(x) = \frac{1}{\log x} \cdot (70 \log\log x)^2$$

such that every $(\frac{1}{2}, q)$-local sparse spanning graph algorithm for the graphs isomorphic to G satisfies $q \geq \log\log(n)/8000$.

4.1 A regular non-expanding graph

The main result in this subsection (stated in Lemma 2) is a construction of regular non-expanding graphs that we will use in Subsection 4.2 to prove Theorem 3. A main ingredient is a result from [29] showing that, roughly speaking, there are graphs that simultaneously have large girth and small hereditary expansion (in fact, small edge separators). While the degree of these graphs may grow with n, their maximum degree is at most a constant times their average degree. We will use this in order to construct a regular graph with similar properties. We note that the regularity condition is crucial for proving Theorem 3. The following theorem was proved in [29].

Theorem 4 ([29])**.** *For any n, k with $2 \leq k \leq \frac{1}{648}\log\log n$ there is an n-vertex graph $G = G_{n,k}$ satisfying:*

i. G has average degree at least k and maximum degree at most $6k$.

ii. G has girth at least $\log n/(6k)^2$.

iii. For every t-vertex subgraph H of G that is not a forest, there exists a subset $S \subseteq V(H)$ of size $(1/3)t \leq |S| \leq (2/3)t$ such that

$$|\partial_H(S)| \leq \frac{t}{\log t} \cdot (\log\log t)^2 \ .$$

We note that each of the parameters in Theorem 3 is quantitatively essentially optimal (see [29] for further discussion).

The main result in this section is the following.

Lemma 2. *For any n_0 there is a connected graph $G_\circ$ on $n \geq n_0$ vertices satisfying:*

i. *$G_\circ$ is 3-regular.*

ii. *$G_\circ$ has girth at least $\log\log(n)/2000$.*

iii. *$G_\circ$ is f-non-expanding with $f(x) = (1/\log x) \cdot (4\log\log x)^2$.*

For the proof we will need the weighted version of the well-known vertex separator theorem for trees. For completeness, we give a short proof below.

Claim 3. *Let $T = (V, E)$ be a tree, and let $w : V \to \mathbb{R}^+$ be a nonnegative weight function over the vertices of T. There is a vertex $v \in V$ whose removal disconnects T into connected components of weight at most $w(V)/2$ each.*[4]

Proof: Start a walk in T from an arbitrary vertex, in each step moving from a vertex u to a neighbor u' if the weight of the tree rooted at u', when the edge (u, u') is removed, is strictly greater than $w(V)/2$. Since T has no cycles and since the walk never reverts the last step taken, the walk eventually stops at some vertex v. This means that when v is removed from T, the weight of the tree rooted at each of the neighbors of v is at most $w(V)/2$. Since these trees are the connected components resulting from the removal of v, we are done. ■

Proof of Lemma 2: Set $k = \log\log m/648$ and let $G_{m,k}$ be the graph from Theorem 4, where we take m to be large enough such that $k \geq \min\{n_0, 2\}$. We note that in the rest of the proof we will use the inequality

$$(6k)^4 \leq \log m \tag{4}$$

which holds since m is sufficiently large. As is well known, by iteratively removing vertices of $G_{m,k}$ of degree at most $k/2$, one obtains a (non-empty) graph of minimum degree at least $k/2$. Let G be a connected component of the largest average degree in the obtained graph. Note that the average degree of G is at least k, the maximum degree is still at most $6k$, and the girth is still at least $\log m/(6k)^2$. Finally, G still satisfies item (iii) of Theorem 4, being a subgraph of $G_{m,k}$.

Let $G_\circ$ be obtained by taking the replacement product of G with a cycle. That is, $G_\circ$ is obtained from G by replacing each vertex of degree x by a cycle on x new vertices – which we henceforth refer to as a "cloud" – and further adding edges as follows: if u, v are adjacent in G, with u being the i-th neighbor of v and v being the j-th neighbor of u (under a fixed arbitrary enumeration of the neighbors of each vertex), then the i-th vertex in the cloud corresponding to v is connected by an edge to the j-th vertex in the cloud corresponding to u. So for example, it is easy to see that there is a one-to-one correspondence between the edges of G and those edges of $G_\circ$ that connect vertices from different clouds. Note that our graph $G_\circ$ is connected, as needed. Letting n denote the number of its vertices, note that n equals the sum of the degrees of all vertices of G, so $n \geq k\,|V(G)| \geq n_0$, as needed. Furthermore, $G_\circ$ is 3-regular, since each vertex has two neighbors in its cloud and one neighbor in precisely one other cloud, as required by item (i) of the statement.

Let us now prove that the girth of $G_\circ$ is equal to the minimum between the girth of G and the minimum degree of G. First, note that any cycle C in $G_\circ$, other than a cloud, naturally determines a closed trail in G (i.e., where vertices may be visited more than once, but not edges). Indeed, for

[4] For a subset $X \subseteq V$ we write $w(X) = \sum_{v \in X} w(v)$.

each edge of C that connects two different clouds, the trail simply moves along the corresponding edge in G.. Note that the length of C is at least the length (i.e., number of edges) of the trail. Since the length of the shortest closed trail in G is its girth, we conclude that the length of any cycle in $G_\circ$ is at least the girth of G, unless that cycle is a cloud. Furthermore, since the smallest number of vertices in a cloud of $G_\circ$ equals the minimum degree of G, our claim follows. That is, the girth of $G_\circ$ is at least

$$\min\{\log m/(6k)^2,\, k/2\} = \log\log(m)/1296 \geq \log\log(n)/2000 \;,$$

where we used the setting of k, Equation (4) and the fact that $n \leq 6k\,|V(G)| \leq 6km \leq m^2$. This proves item (ii) of the statement.

It remains to show that $G_\circ$ satisfies item (iii) of the statement as well. Let $H_\circ$ be a t-vertex subgraph of $G_\circ$. Our goal is to bound $\phi_{H_\circ}$ from above. Let H be the induced subgraph of G obtained by retaining only those vertices whose corresponding cloud has at least one vertex in $H_\circ$. Put $h = |V(H)|$, and notice $t \geq h$. We next consider two cases, depending on whether H is a forest or not.

First, suppose that H is not a forest. Hence, by item (iii) of Theorem 4 (a property which is also satisfied by G, as mentioned above) there is a partition $V(H) = S \cup S'$ with $|S|\,, |S'| \geq h/3$ satisfying $|\partial_H(S)|\,, |\partial_H(S')| \leq (h/\log h)\cdot(\log\log h)^2$. Let $S_\circ$ be the subset of $V(H_\circ)$ corresponding to S (i.e., obtained by replacing each vertex in S with the vertices of its cloud in $H_\circ$). Assume without loss of generality that $|S_\circ| \leq t/2$ (otherwise take $S'_\circ$, which is defined from S' in a similar fashion). Observe that $|\partial_{H_\circ}(S_\circ)| \leq |\partial_H(S)|$, since any edge in $\partial_{H_\circ}(S_\circ)$ must go between two different clouds, and there is a unique edge in $\partial_H(S)$ connecting the two vertices corresponding to these clouds. Therefore,

$$\phi_{H_\circ} \leq \frac{|\partial_{H_\circ}(S_\circ)|}{|S_\circ|} \leq \frac{(h/\log h)\cdot(\log\log h)^2}{h/3} = \frac{3(\log\log h)^2}{\log h} \leq \frac{3(\log\log t)^2}{\log(t/6k)} \leq \frac{6(\log\log t)^2}{\log t} \;,$$

where in the second inequality we used the fact that $|S_\circ| \geq |S| \geq h/3$, in the third inequality we used the fact that $h \leq t \leq 6k \cdot h$, and in the last inequality we used the fact that $t/6k \geq \sqrt{t}$ (i.e., $\sqrt{t} \geq 6k$); the latter follows from the fact that since H is not a forest, h is at least the girth of G, so $t \geq h \geq \log m/(6k)^2 \geq (6k)^2$ by Equation (4). This proves item (iii) of the statement under the assumption that H is not a forest.

Suppose next that H is a forest. Notice we may assume that H is a connected graph since otherwise $H_\circ$ is also not connected, meaning that $\phi_{H_\circ} = 0$ so we are done. We apply Claim 3 on the tree H, where we set the weight of each vertex in H to be the number of vertices in the corresponding cloud in $H_\circ$. Let v be the vertex guaranteed by Claim 3, and let $v_1, \ldots, v_d$ be the vertices of the cloud/cycle corresponding to v, in their order on the cycle. For each $1 \leq i \leq d$, let $S_i \subseteq V(H_\circ)$ be the set of vertices in $H_\circ$ corresponding to the i-th connected components of $H - v$ (i.e., so that v_i is the unique vertex in the cloud of v that is connected to S_i). Put $S'_i = S_i \cup \{v_i\}$. Then $\sum_{i=1}^{d} |S'_i| = t$, and our choice of v guarantees that $|S'_i| \leq t/2 + 1$. We claim that there is an index $1 \leq j \leq d$ such that $(1/4)t \leq \sum_{i=1}^{j} |S'_i| \leq (3/4)t$. Indeed, if $1 \leq j \leq d$ is the smallest index such that $\sum_{i=1}^{j} |S'_i| \geq (1/4)t$ then

$$\sum_{i=1}^{j} \left|S'_i\right| = \sum_{i=1}^{j-1} \left|S'_i\right| + \left|S'_j\right| \leq (t/4 - 1) + (t/2 + 1) = (3/4)t \;.$$

Now, let $S_\circ \subseteq V(H_\circ)$ be the smallest between $\bigcup_{i=1}^{j} S'_i$ and its complement, so that $t/4 \leq |S_\circ| \leq t/2$. Observe that since $\{1, 2, \ldots, j\}$ is an interval, $|\partial_{H_\circ}(S_\circ)| \leq 2$. We conclude that

$$\phi_{H_\circ} \leq 2/(t/4) = 8/t \leq (1/\log t) \cdot (4 \log\log t)^2 \,,$$

where in the last inequality we used the fact that $(x/\log x) \cdot (\log\log x)^2 \geq 1/2$, which can be verified to hold for any real $x \geq 3$ (and thus for any integer $t > 2$). This completes the proof. ■

4.2 Lower bound proof

For our proof of Theorem 3 we will need the graph witnessing the lower bound to contain a bridge. The following lemma shows that one can modify a given graph so as to contain a bridge while preserving high girth and small hereditary expansion.

Lemma 4. *Suppose there is a 3-regular connected n-vertex graph G with girth g that is f-non-expanding, where $f : [1/2, \infty) \to \mathbb{R}$ is monotone decreasing. Then there is a 3-regular connected $(2n+2)$-vertex graph that contains a bridge, and moreover, has girth at least g and is h-non-expanding with $h(x) = 3f(x/2 - 1)$.*

Proof: Let G_1, G_2 be two vertex-disjoint copies of G. Let e_i be an arbitrary edge of G_i, $i \in \{1, 2\}$, and let G'_i be obtained by subdividing e_i. That is, G'_i is obtained from G_i by adding a new vertex w_i, removing the edge $e_i = (u_i, v_i)$ and adding the edges $(u_i, w_i), (w_i, v_i)$. It is clear that subdividing an edge does not decrease the girth. Now, construct the graph F from the union of G'_1 and G'_2 by adding the bridge (w_1, w_2). It is clear that F is 3-regular, connected and has girth at least g. It therefore remains to show that F is h-non-expanding. Let H be a t-vertex subgraph of F with $t > 2$. We need to show that $\phi_H \leq h(t)$. Without loss of generality, H has at least $t/2$ vertices in G'_1. Let H' be the subgraph of H induced by those vertices, where we remove the subdividing vertex w_1 if $w_1 \in V(H)$. Note that H' is a subgraph of G_1. Let $t' \geq t/2 - 1$ denote the number of vertices of H'. Since H' is f-non-expanding, there is a subset $S \subseteq V(H')$ with $|S| \leq t'/2$ and $|\partial_{H'}(S)| / |S| \leq f(t')$. Note that $|\partial_H(S)| \leq |\partial_{H'}(S)| + 2$, since the only edges in H connecting a vertex in H' and a vertex not in H' are (u_1, w_1) and (w_1, v_1). We conclude that

$$\phi_H \leq 3f(t') \leq 3f(t/2 - 1) = h(t) \,,$$

where in the second inequality we used the monotonicity of f for $t \geq 1/2$. ■

For a local sparse spanning graph algorithm $\mathcal{A}$, we denote by $\mathcal{A}(G, u, v) \in \{0, 1\}$ the output of $\mathcal{A}$ when the input graph is $G = (V, E)$ and the input edge is $(u, v) \in E$. The *query-answer transcript* of $\mathcal{A}$ on G, where $\mathcal{A}$ makes q queries and G is d-regular, is the sequence of triples $((x_j, i_j, y_j))_{j=1}^{q}$ where $(x_j, i_j) \in V \times [d]$ is the j-th query and $y_j \in V$ is the corresponding answer.

Finally, for a permutation σ on V, we denote by $\sigma(G)$ the graph isomorphic to G on the same vertex set, for which $(u, v) \in E(\sigma(G))$ if and only if $(\sigma(u), \sigma(v)) \in E(G)$. We stress that in what follows, the graph $\sigma(G)$ will not necessarily have the same neighborhood ordering as that of G. That is, if y is the i-th neighbor of x in G and $\sigma(v) = x, \sigma(u) = y$ then u is *not* necessarily the i-th neighbor of v in $\sigma(G)$.

Lemma 5. *Let G be a 3-regular connected graph of girth g that contains a bridge. Any $(\frac{1}{2}, q)$-local sparse spanning graph algorithm for the graphs isomorphic to G satisfies $q \geq g/2$.*

Proof: Let $\mathcal{A}$ be an $(\frac{1}{2}, q)$-local sparse spanning graph algorithm for the graphs isomorphic to G, and assume, contrary to the claim in the lemma, that $q < g/2$. We shall say that $\mathcal{A}$ *accepts* an edge (u, v) in G if it gives a positive answer when queried on (u, v) (that is, (u, v) belongs to the sparse spanning graph G'). We will show that with probability 1 over its random coins, $\mathcal{A}$ accepts every edge of G. This will complete the proof as it means that the number of edges of G that $\mathcal{A}$ accepts is $(1 + \frac{1}{2})n$, where n is the number of vertices of G, contradicting condition (ii) in Definition 1.

Let $(u, v) \in E(G)$ and assume for contradiction that there is a sequence r of random coins for $\mathcal{A}$ such that the corresponding deterministic algorithm $\mathcal{A}_r$ satisfies $\mathcal{A}_r(G, u, v) = 0$. Suppose, without loss of generality, that the vertex set of G is $[n]$ and that $(1, 2)$ is a bridge in G. We will construct a permutation σ on $[n]$ with $\sigma(u) = 1$, $\sigma(v) = 2$ so that the graph $\sigma(G)$ (with an appropriate way of ordering the neighbors of each vertex) has the property that the query-answer transcript of $\mathcal{A}_r(G, u, v)$ is identical to that of $\mathcal{A}_r(\sigma(G), u, v)$. Note that $\mathcal{A}_r(\sigma(G), u, v)$ is well defined since the input edge (u, v) is indeed an edge of $\sigma(G)$, and since $\sigma(G)$ is a valid input graph to $\mathcal{A}_r$ being isomorphic to G. Since $\mathcal{A}_r$ is deterministic, whether or not $\mathcal{A}_r$ accepts (u, v) depends solely on the query-answer transcript. Therefore, the existence of σ as above would imply that $\mathcal{A}_r(\sigma(G), u, v) = 0$. However, this would contradict condition (i) in Definition 1 since (u, v) is a bridge in $\sigma(G)$.

Let $Q = (x_j, i_j, y_j)_{j=1}^{q}$ be the query-answer transcript of $\mathcal{A}_r(G, u, v)$. We first claim that if a permutation σ and an ordering of the neighbors of each vertex of $\sigma(G)$, are such that $\sigma(u) = 1, \sigma(v) = 2$ and for every $1 \leq j \leq q$ the i_j-th neighbor of vertex x_j in $\sigma(G)$ is vertex y_j then the query-answer transcript of $\mathcal{A}_r(G, u, v)$ is identical to the query-answer transcript of $\mathcal{A}_r(\sigma(G), u, v)$. To see this, let the query-answer transcript of $\mathcal{A}_r(\sigma(G), u, v)$ be denoted by $(x'_j, i'_j, y'_j)_{j=1}^{q'}$. We prove, by induction on j, that the two query-answer transcripts are the same when restricted to the first $1 \leq j \leq q$ queries, that is, $(x'_j, i'_j) = (x_j, i_j)$ and $y'_j = y_j$ for every $1 \leq j \leq q$. Note that this will also imply that $q = q'$ (i.e., that the number of queries is identical). For $j = 1$ we have $(x'_1, i'_1) = (x_1, i_1)$ since $\mathcal{A}_r$ is deterministic and in both cases the input is (u, v). Our assumption on σ thus guarantees that we also have $y'_1 = y_1$. Suppose our claim holds for the first $j - 1$ queries. Again, since $\mathcal{A}_r$ is deterministic, the j-th query is determined only by the query-answer transcript of the first $j - 1$ queries (and the input edge). Hence, the induction hypothesis implies that $(x'_j, i'_j) = (x_j, i_j)$ and our assumption on σ again implies that we also have $y'_j = y_j$. This completes the inductive proof.

It follows that in order to complete the proof it suffices to find a permutation σ and an ordering of the neighbors of each vertex, as above. Let again $Q = (x_j, i_j, y_j)_{j=1}^{q}$ be the query-answer transcript of $\mathcal{A}_r(G, u, v)$, and let F be the (labeled) graph spanned by the edge set[5]

$$E(F) = \{(x_j, y_j)\}_{j=1}^{q} \cup \{(u, v)\} .$$

Since

$$|E(F)| \leq q + 1 \leq g/2 , \tag{5}$$

we have that F is a forest. Let $T_1, \ldots, T_k$ be the (labeled) trees in F. For the sake of defining σ it will be convenient to consider a single tree T. The edge-set of T consists of $E(F)$ and $k - 1$ additional edges. The additional edges do not necessarily belong to G, and are selected as follows.

[5] $E(F)$ might contain the edge (x, y) twice if y is the i-th neighbor of x, x is the j-th neighbor of y and the algorithm queried both (x, i) and (y, j). In this case we will keep just one copy of (x, y) thus making sure that $E(F)$ is indeed a set, and not a multi-set.

For each labeled tree T_i, let t_i denote an arbitrary vertex of degree smaller than 3. For every $i \in [k-1]$, add the edge (t_i, t_{i+1}).

Observe that Equation (5) implies that $|E(T)| < g$. Consider a rooted version of T where u is the root, and construct σ as follows. Set $\sigma(u) = 1$, $\sigma(v) = 2$, and define the neighborhood relation between u, v in $\sigma(G)$ as it is in G. That is, if u is the i-th neighbor of v and v is the j-th neighbor of u in G then the same holds in $\sigma(G)$. Suppose we have already defined $\sigma(x)$ for all x at distance at most $d-1$ from u (in T) as well as for some vertices at distance d, and let y be a vertex at distance d for which $\sigma(y)$ has not been defined yet. Let x be the parent of y in T (whose distance from v is thus $d-1$) and let us set $\sigma(y)$ to be a neighbor of $\sigma(x)$ in G which is not the image of any vertex under the σ we have defined thus far. Such a vertex exists since G is 3-regular and the degree in T is at most 3. If the edge (x, y) is in F then we define the neighborhood relation between $\sigma(x)$ and $\sigma(y)$ as x and y in G. Once we define σ for all vertices of T we arbitrarily extend σ to a permutation, and extend the neighborhood relation between the vertices in a consistent manner. ■

We are now ready to prove Theorem 3.

Proof of Theorem 3: Let

$$h(x) = \frac{1}{\log(3x)} \cdot 32(\log\log(8x))^2 \ .$$

It is not hard to check that $h : [1/2, \infty) \to \mathbb{R}$ is monotone decreasing. Note that the graph in Lemma 2 is h-non-expanding, since for $x \geq 3$,

$$h(x) \geq \frac{1}{\log(x^2)} \cdot 32(\log\log x)^2 = \frac{1}{\log x} \cdot (4\log\log x)^2 \ .$$

Apply Lemma 4 on the graph(s) in Lemma 2. We get a 3-regular connected n-vertex graph, for infinitely many n, that contains a bridge, has girth at least

$$\frac{\log\log((n-1)/2)}{2000} \geq \frac{\log\log(n/4)}{2000} \geq \frac{\log\log(n)}{4000}$$

and is f-non-expanding with

$$f(x) = 3h(x/2-1) \leq \frac{1}{\log(x/2)} \cdot 96(\log\log(4x))^2 \leq \frac{3}{\log x} \cdot 96(4\log\log x)^2 \leq \frac{1}{\log x} \cdot (70\log\log x)^2$$

where we assumed $x \geq 3$. The proof now follows immediately from Lemma 5. ■

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008. 1.1.1

[2] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012. 1.1.1, 1.1.2

[3] N. Alon, P. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the Twenty-Second Annual ACM Symposium on the Theory of Computing (STOC)*, pages 293–299, 1990. 1, 3.1

[4] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S. Teng. Local computation of pagerank contributions. *Internet Mathematics*, 5(1–2):23–45, 2008. 1.1.3

[5] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the Forty-Seventh Annual Symposium on Foundations of Computer Science (FOCS)*, pages 475–486, 2006. 1.1.3

[6] R. Andersen and Y. Peres. Finding sparse cuts locally using evolving sets. In *Proceedings of the Forty-First Annual ACM Symposium on the Theory of Computing (STOC)*, pages 235–244, 2009. 1.1.3

[7] D. A. Bader and G. Cong. A fast, parallel spanning tree algorithm for symmetric multiprocessors (smps). *J. Parallel Distrib. Comput.*, 65(9):994–1006, 2005. 1.1.2

[8] P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006. 1.1.3

[9] Z. Brakerski. Local property restoring. Unpublished manuscript, 2008. 1.1.1

[10] A. Campagna, A. Guo, and R. Rubinfeld. Local reconstructors and tolerant testers for connectivity and diameter. In *Proceedings of the Seventeenth International Workshop on Randomization and Computation (RANDOM)*, pages 411–424, 2013. 1.1.1

[11] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005. 1.1.4

[12] B. Chazelle and C. Seshadhri. Online geometric reconstruction. In *Proceedings of the Twenty-Second Annual ACM Symposium on Computation Geometry (SoCG)*, pages 386 – 394, 2006. 1.1.1

[13] A. Czumaj, A. Shapira, and C. Sohler. Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM Journal on Computing*, 38(6):2499–2510, 2009. 3.1

[14] A. Dutta, R. Levi, D. Ron, and R. Rubinfeld. A simple online competitive adaptation of lempel-ziv compression with efficient random access support. In *Proceedings of the Data Compression Conference (DCC)*, pages 113–122, 2013. 1.1.1

[15] A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of the Fiftieth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2009. 1.1.1

[16] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 271–279, 2003. 1.1.3

[17] M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *Proceedings of the Seventeenth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 433–442, 2011. 1.1.1

[18] S. Kale, Y. Peres, and C. Seshadhri. Noise tolerance of expanders and sublinear expander reconstruction. In *Proceedings of the Forty-Ninth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 719–728, 2008. 1.1.1

[19] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the AMS*, 7(1):48–50, 1956. 1.2, 3.2

[20] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28(1):40–66, 1998. 1.1.2

[21] R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. In *Proceedings of the Eighteenth International Workshop on Randomization and Computation (RANDOM)*, pages 826–842, 2014. $*$, 1

[22] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. 1.1.2

[23] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Computing*, 36(2):177–189, 1979. 1, 3.1

[24] L. Trevisan M. Sudan and S. Vadhan. Pseudorandom generators without the XOR lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing (STOC)*, pages 537–546, 1999. 1.1.1

[25] Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages and Programming: Thirty-Ninth International Colloquium (ICALP)*, pages 653–664, 2012. 1.1.1

[26] Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Proceedings of the Sixteenth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 260–273, 2013. 1.1.1

[27] S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms*, 5(2), 2009. 1.1.1

[28] A. Mayer, S. Naor, and L. Stockmeyer. Local computations on static and dynamic graphs. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems (ISTCS)*, 1995. 1.1.2

[29] G. Moshkovitz and A. Shapira. Decomposing a graph into expanding subgraphs. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015. 4.1, 4, 4.1

[30] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. 1.1.2

[31] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proceedings of the Forty-Ninth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 327–336, 2008. 1.1.1

[32] L. Orecchia and Z. A. Zhu. Flow-based algorithms for local graph clustering. *CoRR*, abs/1307.2855, 2013. 1.1.3

[33] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007. 1.1.1, 1.1.2

[34] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM Journal on Computing*, 30(5):1427–1442, 2000. 1.1.2

[35] S. Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010. 1.1.2

[36] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of The Second Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011. 1.1.1, 1.1.2

[37] M. E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, 39(7):2897–2926, 2010. 1.1.1

[38] T. Sarlos, A. Benczur, K. Csalogany, D. Fogaras, and B. Racz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *Proceedings of the 15th International Conference on WorldWide Web*, pages 297–306, 2006. 1.1.3

[39] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 81–90, 2004. 1.1.3

[40] Y. Yoshida, M. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the Forty-First Annual ACM Symposium on the Theory of Computing (STOC)*, pages 225–234, 2009. 1.1.1

[41] Z. A. Zhu, S. Lattanzi, and V. Mirrokni. A local algorithm for finding well-connected clusters. In *Proceedings of the Thirtieth International Conference on Machine Learning (ICML)*, 2013. 1.1.3